Spring 1-1-2010

Macs: A Practical Approach to Mobile Content Sharing over Ad Hoc Networks

Thomas Kooh
thomaskooh@gmail.com

Macs: A Practical Approach to Mobile Content Sharing over Ad Hoc Networks

by

Thomas Georges Cyrille Kooh

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Computer Science

July 2010

This thesis entitled:
Macs: A Practical Approach to Mobile Content Sharing over Ad Hoc Networks
written by Thomas Georges Cyrille Kooh
has been approved for the Department of Computer Science

_____
Shivakant Mishra, Ph.D., Chair

_____
Kenneth Anderson, Ph.D.

Date_____

The final copy of this thesis has been examined by the signatories, and we
Find that both the content and the form meet acceptable presentation standards
Of scholarly work in the above mentioned discipline.

Kooh, Thomas Georges Cyrille (M.S., Department of Computer Science)

Macs: A Practical Approach to Mobile Content Sharing over Ad Hoc Networks

Thesis directed by Professor Shivakant Mishra

Mobile phones are increasingly equipped with features that allow them to self-generate digital content, and they possess larger storage capacity. With the increased trend of information sharing, promoted by web2.0 applications and the success of peer-to-peer in the wired world, the ability for users to share content on their mobile devices is engaging. Most mobile content sharing solutions work over infrastructure-based networks, such as the internet or the mobile phone network. However, network connectivity is not always available or, at least, affordable. On the other hand, the proliferation of feature-rich mobile devices implies that a mass of digital content can be found nearby. In this context, mobile content sharing applications tailored to ad hoc networks may come in handy for impromptu and ubiquitous sharing. This work studies the feasibility of such solutions through a practical approach, consisting of developing a prototype application. After extensive performance evaluation, it is concluded that ad-hoc-based content sharing is efficiently possible on mobile devices.

Acknowledgments

I wish to acknowledge and thank those without whose help this work could not have been accomplished. Special gratitude goes to my advisor, Professor Shivakant Mishra for his support throughout this work; his guidance and expertise have been of invaluable usefulness; I am also very grateful for him providing me with the mobile devices used for the realization of this work. I cannot refrain from telling my deep appreciation to Professor Qin Lv for providing crucial advice and support since the genesis of this research; she has always been available when needed, even if it took her vacation time. A special thank to Professor Kenneth Anderson for his acceptance to review this work in its last hours. I would also like to tell my sincere gratitude to the faculty and staff of the Department of Computer Science for helping make my graduate program an enjoyable and exciting experience. Words are inadequate to express my thanks to my family – in particular my parents, the Moulema's family, my friends, and Betty Thacker, for their unwavering support and encouragements.

Last but not the least, I acknowledge the unwavering and unshakeable support of The Almighty God, who gave me the strength, energy, and skills necessary to complete this thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

1NF	First Normal Form
2.5G	Second and a half Generation
3G	Third Generation
AP	Access Point
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
HTTP	HyperText Tranfer Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
MANET	Mobile Ad hoc Network
MMS	Multimedia Messaging Service
P2P	Peer to Peer
PC	Personal Computer
PDA	Personal Digital Assistant
RFC	Request For Comments
SIP	Session Initiation Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface

Chapter 1 Introduction

As mobile phones are getting more capable and cheaper, many applications once limited to desktops are now "mobile": digital media players, games, email clients, web browsers, social networking applications, and much more; the range of applications and services available on mobile phones keeps on increasing. This expansion is confirmed by market statistics as Chetan Sharma (2010) forecast a growth of overall mobile applications downloads from 7 billion in 2009 to about 50 billion by 2012. It is also expected that mobile application revenues will exceed $25 billion by 2014 (Juniper Research, 2009). Since applications are all about helping end users to perform specific tasks, these reports from the mobile applications market suggest that people are increasingly interested in doing things that can be done with a PC while on the move.

At the same time, the rapid growth of Peer-to-Peer (P2P) as a computing model has fueled the development of many internet-based applications, such as file sharing, instant messaging, and Voice-over-IP. In a study by Cisco (2008) P2P traffic is expected to remain the lion's share of internet traffic over 2011, quadrupling from 1,330 petabytes per month in 2006 to 5,270 petabytes per month in 2011. One possible explanation of the popularity of P2P systems is that they do not heavily rely on the existence of a dedicated server as it is the case with client/server systems. This characteristic allows P2P systems to be easier to install and maintain, and also to be more resilient to the problem of single-point of failure. In a P2P network, participants (or peers) share resources directly with each other without (or with very limited) central coordination; each peer is able to act as both a provider and a consumer of resources. A particular case is P2P file sharing, an application of P2P in which the resources shared by peers are files. P2P file sharing is by far the most famous application of P2P, generating over 42% of

internet traffic in Northern Africa and 69% of internet traffic in Eastern Europe, as of 2009 (ipoque, 2009).

Although most P2P file sharing applications are PC-based – i.e. designed to run on desktops and laptops, the trend of increased mobility and the proliferation of high-capability mobile phones – sometimes referred to as smartphones or simply mobile phones in this thesis – warrant the development of P2P file sharing systems for mobile devices.

1.1 Research Motivation

Mobile phones are increasingly powerful nowadays and they enable a variety of usage scenarios. As they become feature-rich and possess larger storage capacity, the ability to share user-generated content with others becomes socially attractive (Liu *et al.*, 2009). Along with the success of P2P file sharing in the fixed[1] world, the proliferation of web2.0 applications has contributed to shaping a generation of internet users for which content sharing is not really a revolution. Allowing these users to share the content on their mobile phones while moving is therefore a sound endeavor. However, content sharing on mobile devices is still limited compared to its potential. Most sharing on mobile phones, today, is done through the use of MMS messages, which usually have a size limitation of 300Kb per message. Bluetooth is another widely used method of exchanging digital content among mobile devices; but, support for such exchange on existing mobile phones usually requires a lot of manual configuration from users. Following this observation, some – although very few – mobile P2P file sharing applications have been developed: *Symella* (Kelényi *et al.*, 2007), for instance, is an implementation of a Gnutella peer for Symbian S60 smartphones; *MobileMule* (Emule project) is

[1] "Fixed" here stands for desktops and laptops. Although laptops are portable, we do not consider them as strictly mobile, because it is not very convenient to use one's laptop while on the move. In this perspective, smartphones, for instance, are truly mobile devices - they can be used while walking, shopping, driving, etc.

a mobile application that allows a Java-enabled phone to remotely control an instance of *eMule*[2]; *mbit* (mbit, 2007) is a mobile P2P file sharing software for Symbian S60 and Java-enabled mobile phones, and it uses SIP as its signaling protocol.

All these mobile P2P file sharing applications, like their counterparts on PCs, make at least two assumptions: (1) the underlay network is infrastructure-based and somewhat stable, and (2) the device is connected to the internet. A mobile phone running *Symella* for instance will use *bootstrap* nodes to join the Gnutella network. The implicit design assumption here is that a bootstrap node will easily be reachable, which implies some stability in the underlay network. *mbit* in return, requires that the device be connected to the internet; besides, *mbit* works over the mobile phone operator's cellular network. Mobile P2P content sharing applications that work over infrastructure-based networks, and specifically the Internet, may have the advantage of scalability in that they allow sharing with many users. However, they present some limitations.

First of all, internet-based mobile P2P content sharing applications do not take into account the potential of short-range communication. Bluetooth is available in most mobile phones nowadays, so is Wi-Fi in almost every smartphone; these wireless technologies bring the potential of exchanging files with nearby devices – although this remains a very manual operation in general. An internet-based mobile P2P file sharing solution makes the assumption that content is located on the Internet. However, mobile phones are increasingly equipped with features that not only make them "content consumers", but also "content producers". As an example, every smartphone nowadays is shipped with at least one built-in digital camera, and many phone owners use their mobile devices to take pictures; most mobile phones also have audio and video recording capabilities, which increase the potential of producing self-generated content. Since mobile phones are somehow ubiquitous nowadays, one can conclude that a great

[2] A P2P file sharing application for Microsoft Windows PCs.

deal of digital content is always in our vicinity. In this context, it is not excluded that the file one is looking for is not only on the Internet, but also right next to them.

Another limitation of internet-based mobile content sharing is that internet connectivity is not always a reality in mobile environments; even when it is, usage may imply a cost. Despite the wide deployment of 2.5G and 3G cellular networks, data transfer over these networks is generally not free; therefore, the cost of performing P2P file sharing over the mobile phone network may be a deterrent factor for many users, especially in developing countries. To avoid this cost, an alternative solution could be to use the Wi-Fi interface of the mobile phone to connect to the internet. The problem is that public Wi-Fi coverage is far from being a universal service. In a 6-month study conducted in a US urban area from September 2006 to February 2007, Rahmati and Zhong (2007) found that only 49% of the participants' everyday life was spent under accessible Wi-Fi network[3]. Considering that people generally carry their phone with them, this result implies that one's mobile phone is most of the time unable to connect to the internet via Wi-Fi.

1.2 Objectives and Scope

To address the aforementioned limitations, the main focus of the thesis is to study the feasibility of P2P file sharing applications for mobile devices in ad hoc environments. Specifically, peers are smartphones that collaborate in an infrastructure-less network to exchange files. A typical usage of such mobile P2P content sharing applications is the impromptu sharing of files with people in the vicinity in the absence of internet connection. In this case, phones (or peers) that are within easy reach of each other form a mobile ad hoc network (MANET), and run an overlay protocol to discover and download content. This thesis considers that peers use IEEE

[3] Infrastructure-based Wi-Fi network providing access to the internet.

802.11 (Wi-Fi) as the wireless medium to communicate in the MANET. The preference for 802.11 is motivated by the fact it specifies an ad hoc mode of operation and the fact that smartphones are generally equipped with this technology; besides, Wi-Fi provides longer communication range compared to Bluetooth. In this work, the underlay physical network is assumed to be a single-hop ad hoc network where a mobile phone only interacts with other phones in its wireless coverage, and no peer forwards messages destined to others.

To study the feasibility of the concept, a prototype application is developed, and through a hybrid approach consisting of both simulation and empirical analysis, the effectiveness of the solution is evaluated in terms of: file search speed, file distribution speed, and energy consumption.

The overall aim of this research is to stimulate the development of full-fledged mobile content sharing applications tailored to ad hoc environments. Specifically, within the context of higher education, the objectives of this thesis are to:

- Discuss the relevance of mobile P2P content sharing applications for ad hoc networks.
- Evaluate critically previous works in the area of P2P sharing over MANETs.
- Identify some key features that mobile P2P file sharing applications for MANETs should have, and develop a proof of concept.
- Explore the elements that affect the performance of such applications.

The introductory chapter addressed the first objective by providing some background information, and then by showing the limitations of internet-based content sharing solutions for mobile devices. The remainder of this document addresses the other objectives.

1.3 Outline Structure

After this introductory chapter, **Chapter 2** presents the background knowledge useful to understanding the rest of the thesis. It then explores related works and shows how our work differs from those. The following two chapters constitute the core of the thesis. **Chapter 3** describes the design and implementation of the prototype application. The file discovery and distribution protocols are presented in details, along with a thorough discussion of our design choices. **Chapter 4** is a continuation of the previous one and focuses on assessing the efficiency of P2P file sharing applications for mobile devices in ad hoc environments by using the prototype as a working tool. **Chapter 5** concludes the thesis and introduces potential areas of future work.

Chapter 2 Review of Related Literature

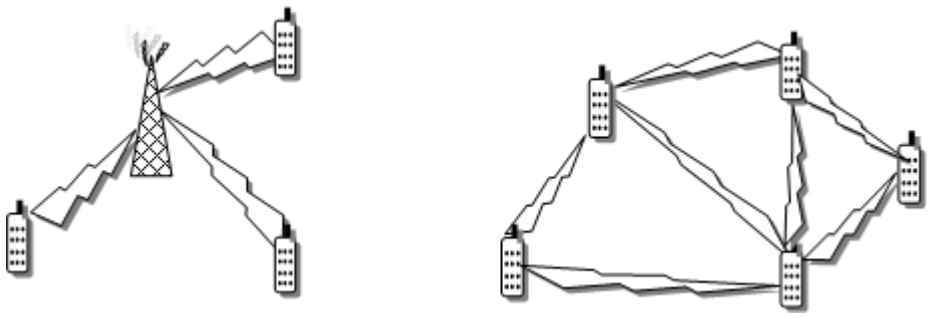
Understanding the rest of this thesis may require some background knowledge. Therefore, relevant concepts are first introduced. After presenting the general concepts, this chapter reviews previous works and clarifies the contribution of our research.

2.1 Background

This section defines some terms and introduces some concepts, namely wireless topologies, and P2P overlay architectures. These concepts are useful to understanding the rest of the document. However, the reader who is familiar with them can skip to section 2.2.

2.1.1 Wireless Topologies

Since mobile communication essentially happens on wireless links, devoting some attention to wireless topologies seems worthwhile. Nodes in a wireless network can typically be interconnected in two ways as shown in Figure 2.1: (a) through a central infrastructure - a special node usually referred to as base station or access point (AP), or (b) through point-to-point links directly connecting one node to another. Although other topologies exist, they are either a repetition of (a) or (b), or a combination of both.



(a) Infrastructure-based (b) Ad hoc

Figure 2-1 Main Wireless topologies

A network is *infrastructure-based* when it includes at least one central infrastructure, whereas an *ad hoc* network is one without any infrastructure – nodes are connected directly to each other. A mobile ad hoc network or MANET is an ad hoc network in which nodes are mobile devices – e.g. Laptops, PDAs, smartphones, etc – (Chlamtac, 2003). However, this thesis considers homogenous MANETs essentially composed of mobile phones.

It is important to note that mobile communication happens in general over infrastructure-based networks. As mentioned by Fitzek *et al.* (2009), this is due in part to the fact that wireless communication is based on radio propagation, which limits the range of device-to-device communication. In this situation, the base station/AP plays the role of a relay point to allow communication between devices afar from each other.

2.1.2 P2P Overlay Architectures

A P2P system can be defined as a distributed system in which participants (or peers) share a part of their resources (processing power, storage capacity, etc.) to provide a service – file sharing in our context; this should be done without (or with very limited) central coordination (Schollmeier, 2002). P2P systems differ from *client/server* systems in that in the latter, each participant is either a service requester (and is called a client) or a service provider

(and is called a server). In P2P systems however, a peer can play both roles; hence the name of "*servent*" given to a peer to express its ability to be both a server and a client (*ibid.*). It is very important to note that a P2P system runs on the application layer. For the system to provide the service, peers communicate with each other forming a virtual network, otherwise called *overlay*. In other words, the P2P system is an overlay network that runs on top of an actual physical network (or underlay) which follows one of the topologies introduced earlier, namely infrastructure-based or ad hoc.

Two important components of any P2P file sharing system are the overlay architecture (the topology of the virtual network) and the mechanism by which the system locates a peer that can serve a specific file. According to these two properties, P2P systems can be classified as *unstructured* and *structured*. Furthermore, unstructured P2P can be divided into *centralized*, *decentralized*, and *hybrid*.

2.1.2.1 Unstructured P2P overlays

In an unstructured P2P system, the overlay is formed arbitrarily (Wang *et al.*, 2003), and the subcategories: *centralized*, *decentralized*, and *hybrid*, denote the process by which a resource is located in the system. For the sake of simplicity and to make the next sections more relevant, we will assume here that resources of interest in the P2P overlay are files.

2.1.2.1.1 Centralized architecture

This architecture uses a special peer, sometimes called super-peer, to index all the files available in the P2P overlay (Figure 2.2). Ordinary peers typically upload meta-information of files they own to the super-peer. A peer *x* requesting a specific file will always query the super-peer. If a match is found, the super-peer sends to peer *x* the address of a peer, say *y*, that actually owns the requested file. Further communications happen directly between peer *x* (the requester)

and peer *y* (the actual file provider). In this architecture, the super-peer is only used as a central directory but it does not own the files.

The vulnerability of the centralized architecture is that it introduces a single point of failure, namely the super-peer; besides, it assumes a certain level of stability in the underlay network[1], which is not characteristic of MANETs. Napster, one of the first P2P music sharing system, followed this architecture (Howe, 2002).

2.1.2.1.2 Decentralized architecture

This architecture, also called the *pure P2P* architecture (*op. cit.*), differs from the previous one in that the P2P overlay here is exclusively composed of ordinary peers; there is no super-peer playing any special role. Instead, all peers are equipotent. A peer *x* requesting a given file will broadcast a query to its neighbors. When a peer *y* receives the query, it responds to the requester if peer *y* owns the file; otherwise, it forwards the query to its neighbors. This process will continue until either a peer responds to the requester or until a set timeout. The pure P2P architecture has the advantage of no single point of failure. It also appears to be more independent on the underlay network because there is no need to contact a central entity (super peer). However, it presents a scalability issue due to the flooding nature of the search. The earliest version of Gnutella (Gnutella v0.4) follows this architecture.

2.1.2.1.3 Hybrid architecture

A combination of the centralized and the decentralized architectures, this architecture addresses the scalability issue of the pure P2P architecture while limiting the single-point-of-failure risk of the centralized architecture. In the hybrid architecture, the P2P overlay is made of many super-peers and many ordinary peers. Each super-peer is responsible for indexing the

[1] A requester peer must always send its query to the super-peer; this assumes that the underlay is stable enough to guarantee that the requester peer will reach the super-peer.

meta-information of files owned by a subset of ordinary peers, and is also connected to other super-peers. When an ordinary peer, say *x*, searches for a file, it will always query its assigned super-peer, say *xx*, as in the centralized architecture. The super-peer *xx* will respond by providing the address of a peer that owns the file if a match is found; otherwise, the super-peer *xx* will forward the query to other super-peers. The process continues until a match is found or a timeout value is reached. If a match is found, further communications happen directly between the requester peer and the provider peer. This architecture has the advantage of being highly scalable, and somewhat resilient to super-peer failure (compared to the centralized architecture). However, it still relies on a relatively stable underlay network because an ordinary peer must always be able to connect to its assigned super-peer in order to start a file discovery. FastTrack (Liang *et al.*, 2006) and the modern Gnutella (Stutzbach *et al.*, 2007) follow this architecture.

To summarize unstructured P2P overlays, Figure 2-2 is provided as a visual representation of the various architectures.

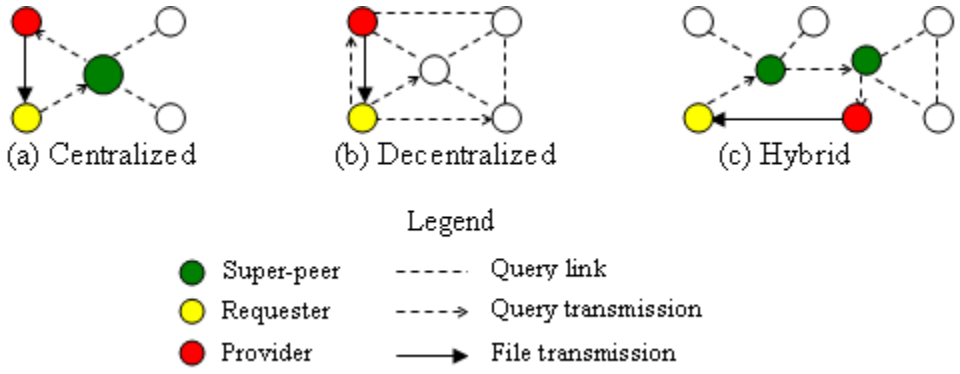


Figure 2-2: Unstructured P2P overlay architectures

2.1.2.2 Structured P2P overlays

A structured P2P system is one in which the formation of the overlay network is strictly controlled so as to make subsequent searches easier to satisfy. A P2P system which follows this

architecture is typically viewed as a distributed hash table (DHT). The idea is to introduce a hash function which always returns values within a known domain, and to distribute the domain of the hash function among all peers in the P2P overlay. Thus, every peer in the overlay knows about at least one copy of each file that hashes within its range, if any. Files are inserted in the overlay by specifying a pair (*key*, *file*); where *key* is the application of the hash function on the file name. Having *key*, the P2P overlay will know on which peer to store the entry (*key*, *file*), since each peer is responsible for a unique range of keys. When a peer *x* searches for a file, it computes the requested file key, say *key_R*, by hashing the name of the file using the known function. Peer *x* then queries the appropriate peer, say *y*, whose key range comprises *key_R*. If an entry (*key_R*, *file*) is found in peer *y*, it means that peer *y* owns the requested file. In that case, peer *y* will respond accordingly to the requester; otherwise the file is considered to be missing.

There are many DHT approaches (Ratnasamy *et al.*, 2001; Stoica *et al.*, 2001; Rowstron *et al.*, 2001; Hildrum *et al.*, 2002); however, one of their limits is that a peer has to know exactly the name (or the key) of a file in order to find it in the overlay network.

2.2 Related work

2.2.1 P2P overlay routing

Mobile P2P content sharing in ad hoc environments is a relatively new area of study and still demands a lot of attention. Nevertheless, most existing research activities have concentrated on the issue of P2P overlay routing for MANETs; in other words given a MANET, how to design the overlay protocol so as to enable efficient file lookup and yet reduce message overhead and redundancy in communication. In a theoretical study, Ding *et al*. (2004) argues for cross-layer approaches. Because the network is very dynamic in MANETs, and since P2P protocols

run at the application layer, cross-layer approaches integrate the network layer with the application layer in order to optimize the routing of messages among peers in the overlay. This solution has also been reclaimed by several other studies. Pucha *et al*. (2004) develops *Ekta*, a system that integrates a structured DHT-based P2P protocol with a MANET multi-hop routing protocol at the network layer; this is done by mapping the IP addresses of the mobile nodes to their node IDs in the DHT namespace. Tang *et al*. (2005) follows the same approach by integrating FastTrack (*op. cit.*) with the AODV routing protocol. Conti *et al*. (2005), on the other hand, presents a cross-layer optimization of Gnutella for MANETs. While these studies are promising, they focus on routing and make the assumption that the underlay network is multi-hop – i.e. every node can route messages. However for this to be true, each node must be configured with a routable IP address, which is far from being guaranteed in real world mobile scenarios. Enabling transparent – with no user configuration – content sharing among mobile phones in ad hoc environments will most likely require the use of link-local IP addresses, which are not routable. Having said this, there is not even a single implementation of a MANET routing protocol on mobile platforms to the best of our knowledge. This also explains why the aforementioned studies lack prototype implementations for mobile platforms, since they tie the P2P application-layer protocol to the network protocol. Instead of following a cross-layer approach, this thesis focuses on the application layer and do not deal with the issue of routing at all.

2.2.2 Data dissemination

Papadopouli *et al*. (2001) presents 7DS, a mobile P2P data sharing system for MANETs. The system defines two modes of operation, namely *prefetch* – where information needs of users are anticipated, and *on-demand* – where information is searched for when a peer fails to access

data via the internet. One limitation of 7DS is that shared data objects must be identified by URLs in order to be discovered by the system. Consequently, data objects are transferred over HTTP, which requires that each mobile device run a web server. However, the impact of running such a server on power consumption is not studied. On the same note, while the authors study the effects of various elements – e.g. wireless coverage – on data dissemination, they omit to evaluate the effectiveness of their data dissemination schemes with increasing object size. Wolfson *et al*. (2007) studies data dissemination on mobile devices with energy, bandwidth, and storage constraints. The authors develop a dissemination algorithm that provides an integral treatment of the three constraints for optimal performance. One of the operations defined in the algorithm, namely *query-response* happens when two peers encounter each other. It consists of having the peers mutually exchange their queries – list of needed files – and eventually receive reports matching the queries. However, the work does not specify the mechanism by which this encounter is done. We assume that it involves the periodic transmission of heartbeat messages, which will bring some communication overhead. Si *et al*. (2009) considers cases in which a file search in the mobile P2P overlay results in multiple potential providers. The study presents a distributed algorithm for selecting one of the providers from which to download the file in such a way that bandwidth is maximized while power consumption is minimized. The proposed algorithm involves the multicast of control messages – ITREQ, ISIMUL, and SIMUL – between the potential file senders and the file requester prior to the file transfer itself. We argue that this introduces some computational load and communication overhead unnecessary in highly dynamic environments such as the one of interest in this thesis.

2.2.3 Power conservation

Energy management should be a very important aspect of mobile P2P file sharing systems for MANETs because the peers are not only battery-driven, but also wireless communication is power-consuming – especially in the case of 802.11. Though the P2P protocol runs at the application layer, various protocols will result in different power consumption. For instance, Kelényi *et al*. (2008) studies the energy consumption of a mobile P2P application that implements a DHT-based protocol. The study finds that phones running the application would only be usable for a couple of hours if they functioned as full peers in the DHT. The authors show that the short operational time is due to the large amount of messages that are exchanged among peers to maintain the DHT. For this reason, we argue for a simplistic overlay protocol that minimizes the number of messages. In general however, energy management strategies are implemented at the link layer and consist of periodically putting the wireless interface to sleep (Zheng *et al*., 2003; Kravets *et al*., 2005).

2.3 Contribution

Instead of only carrying out a theoretical or a simulation-based analysis, our work conducts a feasibility study of ad-hoc-based mobile P2P content sharing by performing – for the most part – empirical analysis based on an actual prototype implementation. The developed prototype implements several file discovery schemes, the most important of which is keyword-based. In the system, users can tag files they share to simplify their discovery; this allows the sharing and discovery of any type of files contrary to a system such as 7DS, which only enables searches for textual files. Besides, the prototype implements advanced keyword-based file search options that bring capabilities that none of the above works support. Specifically, a user can search for:

- Files that match any word in a search expression
- Files that match all words in a search expression
- Files that match exactly a search expression

The prototype also implements a file distribution protocol that is resilient to node failure and connection disruption. The last important aspect of the prototype is the definition of an energy management strategy to maximize the operational time of host mobile phones.

Via extensive measurement, we evaluate the effectiveness of our solution, and show that mobile P2P content sharing for ad hoc networks is definitely feasible with current smartphones.

Chapter 3 Design and Implementation of the Prototype

This chapter gives a description of *Macs*, the mobile P2P content sharing prototype application used throughout this study. Expected features are first identified as a basis for the design. Next, a detailed presentation of the system design is given. The chapter ends with precise information about how *Macs* is implemented.

3.1 Description of Features

The overall function of *Macs* is to allow the exchange of digital content among mobile phones in an ad hoc environment with minimal user interaction. In particular, devices which are in the same vicinity should connect together transparently to form an ad hoc network; that is, no manual user intervention or special configuration server is needed for the devices to connect together. However, actions such as searching for or downloading a specific content may require some intervention from the user.

The following are the key features *Macs* should have; in other words things that the user should be able to do through the application:

- *Share a file*. The user can select any file on their mobile phone and make it public to other devices in the vicinity.
- *Tag a shared file*. The user can choose to tag a shared file to make related searches in the network easy to satisfy. A tag is a word or expression that describes the shared content.
- *Stop sharing a file*. The user can choose to make private a file that was previously shared.
- *List shared files*. The user can view the list of shared files on their phone.

- *Search for a file*. The user can search for a file in the network either through the use of keywords – terms that describe the needed file, or by specifying a predefined search criterion – e.g. the most recent shared file, or the most downloaded file. The search may provide no result when no matching file is found in the network.
- *Save a search*. When a search is not satisfactory, the user can choose to save it for later. In that case, *Macs* will periodically search for that file and notify the user when it is found.
- *Cancel a saved search*. The user can delete a previously saved search.
- *Download a file*. The user can download a discovered file from the network. The download may not complete in a single attempt; therefore the application should allow partial downloads to resume automatically.

A graphical representation of *Macs*' features is given in the use case diagram of Figure 3-1.

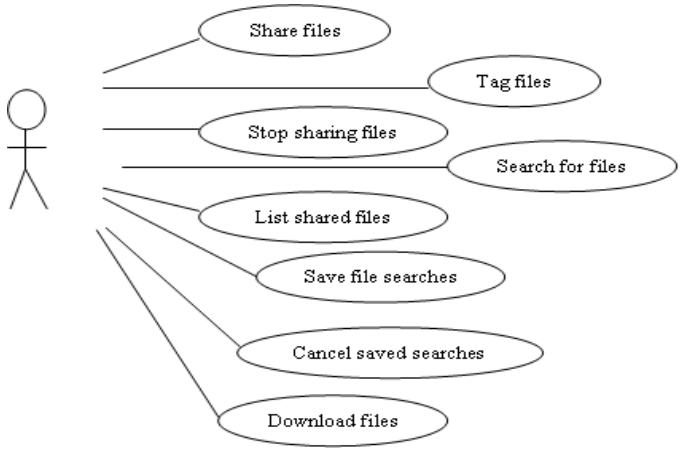


Figure 3-1: Use case diagram of Macs

From the above description, many scenarios in which *Macs* may come in handy are imaginable. Here is a simple example: Lisa, a recent graduate, is riding the bus on her way to work. For some reason, she suddenly feels bored and decides to use our application to search for

the latest pictures taken by other people on the bus; at least that way she can have some fun before the bus reaches her destination. So she brings up the application and searches for files matching the expression '*mountain pictures*'. Fortunately, one of the bus riders, who just returned from climbing Mount Kilimanjaro last weekend, happens to have shared on his smartphone some nature pictures taken from the top of the mountain that he tagged '*beautiful mountain*'. So, Lisa is pleased to discover that her search results in some hits. Based on the meta-information of matched files (size, additional tags, date, etc.), she decides to download a couple on her own mobile phone and enjoys looking at them.

Notice that in the previous example, Lisa is not aware of the devices from where the content is downloaded. Her access to the network is totally transparent and does not require any prior configuration. The only thing she has to do is take her phone and enter her search expression. Similarly, no intervention is requested from the owner of the phone which provides the content at the time Lisa is downloading it. The only thing he had to do was to take the pictures, tag them, and share them.

3.2 System Design

3.2.1 Overview

As a P2P system, *Macs* is distributed, which means that the service is provided through the cooperation of multiple peers. In the following, the term *peer* refers to a mobile phone running an instance of *Macs*. Specifically, we model *Macs* as an unstructured, decentralized mobile P2P system (see section 2.1.2.1). In other words, there is no such concept as central peers or ultra-peers; instead, all peers are equipotent and play the same role. Such a pure P2P model is accurate because *Macs* captures the interaction among mobile phones in an ad hoc environment;

thus, the transience of the underlay network will render any attempt to connect to a central peer needless if not impossible. For the same reason, *Macs* does not specify an explicit mechanism to form or maintain the overlay network. In other P2P systems, peers exchange heartbeat messages to actively discover other peers in the overlay network or to join the overlay. In *Macs*, however, a peer implicitly (or passively) discovers its neighbors as it searches for content.

While some works address the issue of routing in MANETs, we are not aware of any implementation for mobile phones. *Macs* targets the application layer and do not deal with the issue of routing at all. Consequently, the underlay is a single-hop ad hoc network where a peer only interacts with other peers in its wireless coverage, and no peer forwards messages destined to others.

3.2.2 Network Address Assignment

Each mobile phone running *Macs* must have a unique IP address assigned to it in order to participate in the P2P overlay. A manual IP configuration is prohibited according to our requirements because it will necessitate the user intervention. Besides, a manual configuration is not even feasible because the user would have to know the IP addresses of other peers in the vicinity to avoid duplicates, and ensure that the assigned IP address has the same network prefix as the addresses currently used by other peers. On the other hand, the ad hoc nature of the underlay network implies the absence of a DHCP server for dynamic IP configuration. Therefore, *Macs* uses link-local addressing as specified in RFC 3927 (Cheshire *et al.*, 2005); a peer is automatically assigned an IPv4 address in the special range 169.254/16. One of the characteristics of link-local addresses is that they are not routable according to the specification, which reinforces our focus on single-hop communication.

3.2.3 Content organization

In the following, the terms *content* and *file* are used interchangeably. However, content can also refer to the information or data contained in a file. The exact meaning of one term or the other will be obvious from the context in which we use it.

The logical organization of files in *Macs* is fundamental to both how content is discovered and how it is distributed in the overlay network. Each file has a unique identifier or hash value, which is not the name of the file. This identifier is actually computed by applying a cryptographic hash function on the file data; therefore, two files with the same content are considered identical no matter what names they have. Files may also have one or more tags attached to them. These tags play an important role in the discovery of files in the network, as explained later. Because *Macs* operates in transient environments, each file is logically split into fixed-size pieces, called *chunks* – eventually the last chunk of a file can be smaller than the standard chunk size. To guarantee the correctness of file transfer, a hash is also computed for each chunk allowing the receiver of a chunk to verify its integrity. The logical structure of a file is given in Figure 3-2. As one can see, some chunks are shaded to illustrate the fact that a shared file can be incomplete on a peer.

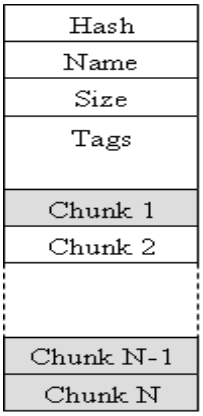


Figure 3-2: Logical organization of a file in Macs.
Shaded boxes denote missing chunks

3.2.4 Content Discovery Protocol

The *Macs* content discovery protocol defines the ways in which peers interact over the overlay network to discover, or to find, files. It specifies a set of messages[1] used for communication between peers and a logic governing the inter-peer exchange of messages. Although *Macs* is a pure P2P system, we use some terms for the sake of clarity: *client peer* refers to a peer benefiting from the service of other peers, and *server peer* refers to a peer providing the service to other peers. Of course, a peer can be involved in two communication sessions simultaneously, acting as both a client and a server peer. There are three ways to find a file in *Macs*: (1) descriptive discovery, (2) identity-based discovery, and (3) opportunistic discovery.

3.2.4.1 Descriptive Discovery

This is a keyword-based file discovery. In this scheme, a peer discovers a file in the network by broadcasting a query describing the needed file. Typically, the description consists of a search expression supplied by the user. The following table defines the control messages used for this type of discovery. The table lists these messages in the order in which they would be exchanged in a normal usage scenario.

Table 3-1: Messages used for descriptive content discovery

Message	Description	Communication method	Protocol
FileSearch	Starts the discovery process. Broadcast by a client peer when searching for a file. The message embeds a search expression, and its purpose is to query the network for	Broadcast to all peers	UDP

[1] The formats of all messages used in *Macs* are provided in Appendix A.

	files that match the search expression.		
QueryHit	The response to a “FileSearch”. Only sent by a server peer if it finds one or more local shared files that match the expression specified in the previously received “FileSearch”. The “QueryHit” message embeds a reference number later used by the requester peer to retrieve the metadata of matched files.	Unicast to the requester peer from which this peer received a “FileSearch”	UDP
MetadataRequest	The follow-up to a “QueryHit”. Sent by a client peer to each server peer that issued a “QueryHit” message. This denotes the client peer’s intent to receive the metadata of matched files. This message embeds the reference number contained in a previously received “QueryHit”.	Unicast to each peer from which this peer received a “QueryHit”	UDP
InitMetadataTransfer	On receiving a “MetadataRequest”, a server peer reads the embedded reference number to determine which shared files are concerned. It then sends the “InitMetadataTransfer” message to initiate the metadata transfer of those files. This message contains the TCP port to which the client peer must connect to download the metadata.	Unicast to the peer from which this peer received a “MetadatRequest ”	UDP

The main goal of the descriptive content discovery is to find files in the network that match a search expression – if any, and to provide the requester peer with metadata of those files. Pieces of information composing a file metadata are: the file hash, its sharing name, its size, its

type, its tags, its sharing date, the number of times it was downloaded, and its relevance. The metadata helps the user on the requester peer to choose which file to download in the case of multiple matched files. At the end of the descriptive discovery, a peer will also know addresses of all peers in the network that possess a given matched file. This information is stored in a temporary hash table, called *PROVIDER_LIST*. This table is indexed by file hash so that a call to *PROVIDER_LIST[hvalue]* will return the IP addresses of all peers that possess the file with hash *hvalue*. In actuality, an element of *PROVIDER_LIST* is a structure defined as follows:

```
Struct Provider {
  Integer provider_addr;
 String available_chunks;
};
```

Where *provider_addr* is the IP address of the provider peer, and *available_chunks* is a bit string denoting which chunks are available on the provider peer. For example, if the file in question has four chunks, a value of “0011” would mean that only the last two chunks are available.

An important restriction by the protocol is that a file has to be in a complete state (without missing chunks) on a peer in order to be considered a possibly matching file. Therefore, each entry in *PROVIDER_LIST* added as a result of the descriptive discovery will have its *available_chunks* field set to all “1”.

The “InitMetadataTransfer” message initiates a TCP connection. On receiving “InitMetadataTransfer”, a client peer will connect to the TCP port specified in the message, and then the metadata download of matched files will follow. In other words, the actual metadata transfer happens over TCP instead of UDP. This is because a file metadata comprises sensitive

pieces of information that should be exchanged reliably. Metadata is not transferred over HTTP; instead raw TCP sockets are used for simplicity. HTTP would require a running instance of an HTTP server on each peer, which would bring additional complexity to the system.

3.2.4.1.1 Query Matching

An important aspect of the descriptive content discovery is the query matching performed by a server peer after reception of a "FileSearch" message and before the send of a "QueryHit" message. The purpose of the query matching is to return the list of shared files on a peer that match the search expression specified within a received "FileSearch" message. The query matching consists of matching the search expression in a "FileSearch" message to tags of shared files. Formally, let *F1* denote a shared file on a peer, and (tag_1, tag_2,…, tag_N) the collection of tags associated to *F1*. Furthermore, let *expression* = $word_1$ $word_2$ … $word_M$ be the search expression embedded in a "FileSearch" message – the search expression may be composed of multiple words. We consider that *F1* matches the "FileSearch" if one of the following conditions is met:

i. $\exists i \in [1, M]$, such that $word_i \in (tag_1, tag_2, ..., tag_N)$

ii. $\forall i \in [1, M], word_i \in (tag_1, tag_2, ..., tag_N)$

iii. $\exists i \in [1, N]$, such that tag_i = *expression*

The first condition (i) is true when any word in the search expression matches at least one file tag; the second condition (ii) when all words in the search expression match at least one file tag; and the third condition (iii) when there is at least one file tag that exactly matches the whole search expression. The matching condition to meet is specified in the "FileSearch" message (see format of "FileSearch" message in Appendix A).

3.2.4.1.2 Ranking of matched files

Each matched file according to the above algorithm is also associated a rank, which denotes the relevance of the file with respect to a received "FileSearch" message. The rank is simply the number of words in the search expression that were matched. This implies that all words in a search expression are checked even if the "FileSearch" message requested an "any word" type of match (condition i. above). For exact matches (condition iii. above), the rank is the maximum value.

We conclude the presentation of the descriptive discovery scheme by giving general algorithms modeling the inter-peer exchange of messages. We use the generic variable *fhash* to refer to the hash of a file.

Client peer

```
[Broadcast FileSearch]
While ([Receive QueryHit] AND (Not timeout))
    [Send MetadataRequest] to sender of QueryHit
    If ([Receive InitMetadataTransfer]) Then
      Connect to sender's TCP port
      Download file metadata
      Add metadata sender to PROVIDER_LIST[fhash]
    End if
End while
```

Server peer

```
If ([Receive FileSearch]) Then
  Run [query matching]
```

```
    If (match found) Then
      Cache metadata of matched files
      [Send QueryHit]
    End if
  End if
  If ([Receive MetadataRequest]) Then
    Open TCP port
    [Send InitMetadataTransfer] to sender of MetadataRequest
    Wait for incoming TCP connection
    If (incoming TCP connection) Then
      Upload metadata to connected peer
    Else
      Timeout
    End if
  End if
```

3.2.4.2 Identity-based Discovery

The key difference between this type of discovery and the descriptive discovery is that in the latter, the peer does not know the identifier of the file searched for; instead, a search expression is supplied by the user to find the file. In the identity-based content discovery, the peer knows the hash of the file searched for, hence the term “identity-based”. This type of content discovery is always performed automatically by the peer without the user intervention. An identity-based discovery is triggered on failure of a file download (we will cover file download later). When the download of a given file fails, the downloading peer will lack some

chunks of the file in question and that file will be in an incomplete state on the peer. The goal of the identity-based discovery is to find any peer in the network that can provide at least one of the missing file chunks. When such peer is found, a new entry is added to the hash table *PROVIDER_LIST*. In order words, the identity-based discovery serves the purpose of filling in the table *PROVIDER_LIST* so as to prepare subsequent downloads. The following table defines the messages used for this type of discovery. Once again, the messages are listed in the order in which they would be exchanged in a typical scenario.

Table 3-2: Messages used for identity-based content discovery

Message	**Description**	**Communication method**	**Protocol**
FileProbe	Starts the discovery process. Broadcast by a peer to probe the network for a specific file. The hash of the needed file is embedded, as well as the indexes of missing chunks.	Broadcast to all peers	UDP
FileProbeHit	The response to a "FileProbe". Only sent by a peer if it happens to possess the probed file and at least one of the chunks needed by the requester peer. The "FileProbeHit" message embeds the hash of the file in question, as well as indexes of actual chunks that the server peer can provide.	Unicast to the peer from which this peer received a "FileProbe"	UDP

Through the identity-based content discovery, a peer having an incomplete file – typically due to a previously interrupted download – can detect the existence of missing chunks in the network. This discovery is generally followed by the download of the actual file chunks

(this will be covered in the section dealing with content distribution). An important note is that a peer does not need to have the complete file to respond to a “FileProbe”. Instead, a peer may send a “FileProbeHit” as long as it can provide at least one chunk needed by the requester peer.

The general algorithms defining the inter-peer exchange of messages for the identity-based content discovery are given below. We use the generic variable *fhash* to refer to the hash of the file causing the execution of the discovery.

Client peer

```
While (PROVIDER_LIST[fhash] is empty)
  [Broadcast FileProbe]
  If ([Receive FileProbeHit]) Then
    Add sender of FileProbeHit to PROVIDER_LIST[fhash]
  Else
    Sleep for some time
  End if
End while
```

Server peer

```
If ([Receive FileProbe]) Then
  If (probed file exists) AND (chunks available) Then
      [Send FileProbeHit] to sender of FileProbe
  End if
End if
```

3.2.4.3 Opportunistic Discovery

The opportunistic discovery is similar to the identity-based discovery in that it allows a peer to detect the existence of missing chunks in the overlay network. However, the opportunistic discovery is triggered by the reception of a report from a peer that just completed the download of a file; it is therefore a passive discovery process.

When a peer completes the download of a file, it will broadcast a “FileReport” message to inform other peers in the vicinity of the availability of the recently obtained file. The file hash is included in the “FileReport” message. On receiving the message, a peer still missing some chunks of the referenced file will detect the existence of a provider. Download of the missing chunks can eventually follow (but this is part of the content distribution protocol described later). Note that a “FileReport” message is sent by a peer that completed the download of a given file. Therefore, a peer sending this message possesses the totality of the file chunks.

The general algorithms used in this type of discovery are given in the following. We use the generic variable *fhash* to refer to the hash of the file causing the execution of the discovery.

Client peer

```
If ([Receive FileReport]) Then
  If (referenced file is incomplete) Then
    Add sender of FileReport to PROVIDER_LIST[fhash]
  End if
End if
```

Server peer

```
If (file download completed) Then
```

[Broadcast FileReport]

End if

3.2.5 Content Distribution Protocol

While the content discovery protocol specifies the ways in which files are found in the network, the content distribution protocol specifies the way in which they are distributed from peer to peer. Files are distributed in chunks, which makes a chunk the smallest transferable unit of file data. This allows a peer to download a given file in multiple attempts and from different providers, which is certainly desirable in mobile environments. A peer can start downloading a specific file from peer *x*, and complete the download from peer *y* thirty minutes later. When all chunks of a given file are received without corruption, the file is considered to be completely downloaded. The transfer of a file involves the exchange of control messages between the client peer and the server peer. This control messages are exchanged even before the actual chunks transfer occurs. The following table gives a description of the messages, which are listed in the order in which they would be exchanged in a typical scenario.

Table 3-3: Control messages used for file transfer

Message	Description	Communication method	Protocol
FileDataRequest	Sent by a client peer to request a file transfer. The file hash is embedded in the message as well as the indexes of the needed file chunks. Note that at the time of sending this message, a peer knows that the destination peer certainly possesses the chunks to download.	Unicast to a server peer	UDP
ReadyToSendFile	The response to a "FileDataRequest". Sent by a server peer to a client peer to prepare a file transfer. This message contains the TCP port to which the client peer must connect in order to download the file.	Unicast to the peer from which this server peer received a "FileDataRequest"	UDP

3.2.5.1 Provider selection

The download of a file is always subsequent to its discovery. Therefore, at the time of sending a "FileDataRequest", a client peer already has at least one provider in the hash table *PROVIDER_LIST*. Having multiple providers for a file is not a necessary condition for its download; but when there are multiple providers, a client peer will always start sending the "FileDataRequest" to the provider with the most number of chunks. When there is more than one such provider, the choice of the one to start with is random. This selection scheme is possible because all potential providers are stored in the table *PROVIDER_LIST*. Besides, each element of the table has a field *available_chunks* (see definition of the structure in section 3.2.4.1), which makes it possible to determine the number of chunks the corresponding peer can provide. When

a client peer sends the “FileDataRequest”, it activates a timer. If the peer receives a “ReadyToSendFile” before the timer pops, the timer is stopped and the chunks download starts. However, if the peer does not receive a “ReadyToSendFile” before the timer expires, the peer to which the “FileDataRequest” was sent is deleted from *PROVIDER_LIST* and the process restarts with the next provider in the list that has the most number of chunks. When *PROVIDER_LIST* becomes empty, the identity-based discovery is executed as explained earlier in section 3.2.4.2 of this document. The following algorithms illustrate the selection of a file provider. We use the generic variable *fhash* to refer to the hash of the file to download.

Client peer

```
While (PROVIDER_LIST[fhash] is NOT empty)
  SelectedProvider =
        GetProviderWithMostNumberOfChunks(PROVIDER_LIST[fhash])
  [Send FileDataRequest] to SelectedProvider
  [Wait ReadyToSendFile]
  If (Timeout) Then
    DeletedFrom(PROVIDER_LIST[fhash], SelectedProvider)
  Else
    Connect to TCP port
    Download chunks from SelectedProvider
  End if
End while
```

Server peer

```
If ([Receive FileDataRequest]) Then
  Open TCP port
  [Send ReadyToSendFile] to sender of FileDataRequest
  Wait for incoming TCP connection
  If (incoming TCP connection) Then
    Upload chunks to connected peer
  Else
    Timeout
  End if
End if
```

3.2.5.2 File integrity and Download Resumption

As stated earlier, files are transferred by chunks. However, *Macs* does not use HTTP to transfer chunks, instead the transfer is performed via raw TCP sockets in two steps as follows:

i. First, the server peer sends hashes of all chunks to be transferred.

ii. After step (i), the server peer sends the actual chunks (pieces of the file data), one at a time. The chunks are sent in random order to increase the opportunity of exchanging data among peers.

On receiving a chunk, a client peer determines the success of the transfer by computing the hash of the received chunk and comparing it with the hash received as a result of step (i). If the two hashes match, the integrity of the chunk is verified. Otherwise, that chunk is considered to be corrupted and still missing, which will result in an incomplete file download. Another cause of incomplete file download is the TCP connection break which may result from the peers' mobility.

In any case, an incomplete download triggers the execution of the download resumption. Resuming a download is in fact identical to the provider selection algorithm presented in the previous section. If a download does not complete successfully, the downloading peer will execute the same algorithm described in section 3.2.5.1 above. Once all chunks are successfully received, the receiver peer considers the file as complete and automatically broadcasts a "FileReport" message (see section 3.2.4.3).

3.2.5.3 Simultaneous upload/download

Note well that since files are organized and transferred in chunks, a peer can simultaneously upload chunks of a given file to another peer as it downloads other chunks of the same file from another peer; an illustration of this is given in Figure 3-3. In Chapter 3 – Performance Evaluation and Results – we explore the benefit of simultaneous download/upload.

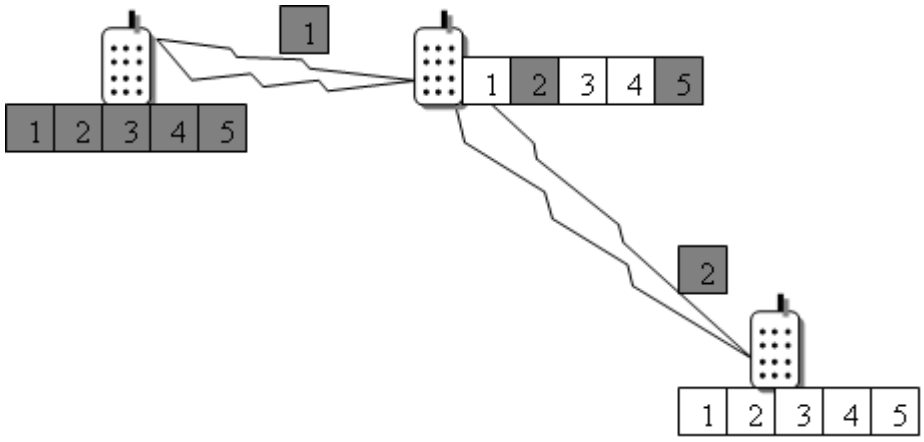


Figure 3-3: Illustration of simultaneous upload/download

The shaded boxes represent chunks available on a peer. The peer on the left has all the file chunks available. The peer in the middle is downloading chunk 1 from the peer on the left. Simultaneously, the peer in the middle is uploading chunk 2 of the same file to the peer at the bottom-right (which happens to be out of the wireless coverage of the peer on the left).

3.2.6 Energy Management

Energy consumption of battery-driven devices is a crucial concern for users. The longer the operational time of the device, the better the user's experience. Taking this fact into consideration in the design is therefore appropriate, especially since *Macs* is essentially a network application and previous works already established the high energy consumption of wireless communication in mobile devices (Balasubramanian *et al.*, 2009; Rahmati *et al.*, 2007; Flinn *et al.*, 1999). A naïve design would consist of keeping the wireless interface of a peer always turned on to ensure all incoming messages are received and processed appropriately. However, this would imply significant energy consumption due to the high cost of maintaining Wi-Fi interfaces up. Therefore, *Macs* adopts an adaptive duty cycling approach, which consists of alternating a peer between *awake* and *sleep* modes periodically. A peer in *awake* mode has its wireless interface turned on, and can participate in the overlay network. On the other hand, a peer in *sleep* mode cannot communicate in the network because its wireless interface is turned off. Therefore, the length of time spent in either mode has an impact on the efficiency of the overlay network. Spending a long time in *awake* mode will result in high responsive peers, but also high energy consumption and short operational time. On the other hand, spending a long time in *sleep* mode will result in low responsive peers, but also low energy consumption and long operational time. So, there is a trade off between efficiency and energy. In *Macs*, a peer does not blindly switch from *awake* mode to *sleep* mode; instead, the transition is based on previous observations. The assumption is that if a peer receives an incoming message at time t_0, it is likely that it will receive another one at time t_1; thus, *Macs* will keep the peer *awake* for time t_1.

We now give a formal description of the energy management in *Macs*. The execution time is divided into time periods (or time intervals) of equal length. For a time period T_i, a peer is either in *awake* mode or in *sleep* mode. The problem is to determine which mode the peer will be in for the next time period T_{i+1}. This decision is made as follows:

(a) A peer always starts in *awake* mode; thus, for T_0 a peer is always *awake*.

(b) If a peer was in *sleep* during T_i, always switch the peer to *awake* for T_{i+1}.

(c) If a peer was *awake* during T_i, switch the peer to *sleep* for T_{i+1} only if

i. No message was received by the peer during T_i,

OR

ii. The peer had already been *awake* for *MAX_AWAKE* consecutive time periods.

The following figure is a graphical illustration of the energy management used in *Macs*. In the figure, the high energy value means that the peer is *awake*; the *sleep* mode is represented by the low energy value. The dots on the energy curve at T_2, T_3, and T_4 denote received messages. Finally, *MAX_AWAKE* = 3 in this illustration.

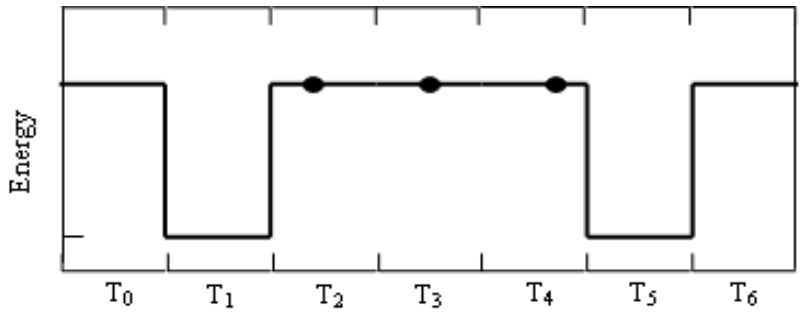


Figure 3-4: Illustration of energy management in Macs
Transitions of a peer between *awake* and *sleep* modes.

3.3 System Implementation

Macs consists of six components, each delivering one or more functions necessary to meet the features identified in section 3.1. The components, represented in Figure 3.5, are not isolated pieces but they interact with one another by invoking each other's services.

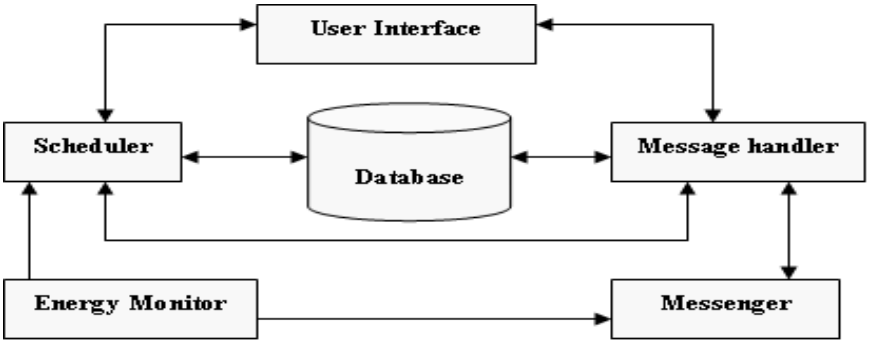


Figure 3-5: Software components of Macs

Macs is currently implemented on Symbian S60 3rd edition FP1 devices (Nokia, 2006). We specifically use Qt for Symbian version 4.6 (by Nokia) as our development and UI framework, coupled with SQL for the database engine. In the following, we describe each component of the application.

3.3.1 Database

The database is implemented in SQLite, an in-process, software library that models an SQL-based relational database engine. We use the version embedded in the "Qt for Symbian" package. The purpose of the database is to logically organize shared files on a peer. It essentially consists of three relations as shown below:

Table 3-4: SharedContent relation

Column	*Type*	*Description*
Content_id	Integer	Primary key
Hash	Varchar(20)	File hash
Path	Varchar(75)	File path on the local file system
Alias	Varchar(150)	Sharing name. This is the name seen by other peers
Type	Integer	File type. Currently support: unknown(0), audio(1), image(2), and video(3)
Shared_date	Integer	Date the file was shared
Downloads	Integer	Number of times the file was downloaded
Size	Interger	The file size
Tags	Varchar(1000)	Comma-separated list of file tags

Table 3-5: Chunk relation

Column	*Type*	*Description*
Chunk_id	Integer	Primary key
Chunk_index	Integer	The index of a file chunk
Hash	Varchar(20)	The chunk hash
Downloaded	Integer	1: the chunk has been downloaded; 0: the chunk is missing.
Ref	Integer	Foreign key referencing column “Content_id” of relation “SharedContent”

Table 3-6: LoggedSearch[2] relation

Column	*Type*	*Description*
Search_id	Integer	Primary key
Search_term	Varchar(250)	The search term

Strictly speaking, the above database design is not compliant with the 1NF of database normalization. The relation *SharedContent* contains a column with non atomic values, namely *Tags*. This column stores the list of tags associated to a shared file. A 1NF-compliant design would consist of removing that column from the "SharedContent" relation and creates two new relations: One that will store single tag strings, say *TagString*, and another that will link the current "SharedContent" relation with "TagString", say *Tagged_content*. Such a design would bring the benefit of not duplicating tag values; however, it would require a joined query to return the list of tags associated to a file. Join queries are complex operations that require processing power that current mobile phones may not provide. In fact, in Chapter 4 we show the performance gain of not using the 1NF-compliant design.

3.3.2 Messenger

This component is simply the application's interface to the overlay network. Running as a thread, it is the communication layer responsible for sending and receiving messages. All values are transferred over the wireless medium in a defined binary format. The "Messenger" component is responsible for making conversions from (respectively, to) the binary format. The following specifies the binary encoding of supported types.

Integers encoded in big endian.

Characters encoded in UTF-8.

[2] This relation stores file searches for later processing, as defined earlier in the document.

Strings	encoded in UTF-8, prepended with an encoded integer denoting the length of the string.

The “Messenger” component also implements a queue through which it receives messages to be sent out from the “Message Handler” component. Similarly, when “Messenger” receives a message from the network, it passes it to “Message Handler” by putting it to a queue owned by “Message Handler”.

3.3.3 Message Handler

Implemented as a thread, its main purpose is to process incoming and outgoing messages as dictated by the content discovery and the content distribution protocols. It processes file searches and issues responses. The query matching (as defined in section 3.2.4.1.1) is performed by this component. To find which files match the search expression specified in a received “FileSearch” message, the “Message Handler” component submits an SQL query to the database requesting the list of files whose tags include any, or all words in the search expression. Each row in the list of files returned by the database has the following columns (all taken from the “SharedContent” relation):

Hash | Alias | Type | Size | Shared_date | Downloads | Tags | rank[3]

Once “Message Handler” receives the list of files from the database (in case there are any matched files), it saves the list to a temporary file on disk and forms the “QueryHit” message; this message is sent as a response to the previously received “FileSearch”. Later on when a

[3] Computed at query time by the database engine (see section 3.2.4.1.2)

corresponding “MetadataRequest” message is received, the “Message Handler” component will transfer the items in the temporary file to the requester peer.

All outgoing message is formed by this component and simply passed to the “Messenger” for effective transmission over the network.

3.3.4 Scheduler

The scheduler is responsible for performing automatic (non-user initiated) tasks, namely executing saved file searches and resuming failed downloads. After every defined period of time, the scheduler will check the database to determine whether a task should be executed. For saved file searches, it checks the “LoggedSeach” table to see whether it has any row. In the positive case, it issues an appropriate “FileSearch” message corresponding to the table entry. Of course, the scheduler itself does not form the message; this is done in “Message handler”. For failed – or incomplete downloads, the scheduler checks the database for files that have missing chunks (i.e. attribute “Downloaded” of table “Chunk”). It then contacts the “Message handler” to send out the appropriate message if such a file is found.

The scheduler execution frequency is a parameter of the application modifiable by the user. By default the value of that parameter is one minute.

3.3.5 Energy Monitor

This component implements the energy management algorithm as defined earlier in section 3.2.6. It switches a peer to *sleep* mode by disabling the “Messenger” and the “Scheduler” components. The peer is put in *awake* by enabling the same two components.

3.3.6 User Interface

It implements the graphical interface that users manipulate to interact with the application. The interface may also update the database depending on the actions of the user, hence its relation with the database as represented in Figure 3-6. The following figure is a snapshot of the application user interface.

Figure 3-6: Sample screenshots of *Macs*

Chapter 4 Performance Evaluation

Central to our study is the evaluation of the feasibility of mobile P2P file sharing applications for ad hoc networks. A necessary step was to implement a prototype application, which was described in the previous chapter. This chapter focuses on assessing the efficiency of such solutions by using the prototype as a working tool. Although we use a specific implementation, the conducted experiments concentrate on aspects which we believe are characteristic of all mobile P2P file sharing applications, namely the search speed, the energy consumption, and the efficiency of file distribution. The main goal of these experiments is to identify factors/elements that influence the application performance. We start by describing our measurement methodology. Then, we show the experiments results and discuss each of them.

4.1 Measurement Methodology

As mentioned above, we measure the search speed, the energy consumption, and the efficiency of file distribution of *Macs*. For the first two aspects we use observations from running the prototype on actual mobile devices. We deploy *Macs* on two Nokia N95 devices all running Symbian OS v9.2. Each device is equipped with a WLAN 802.11b interface, two ARM-11 processors at 332 MHz CPU clock rate, and 18MB of executable RAM. Furthermore, each device is equipped with a 1GB micro SD card for secondary storage and powered by a BL-5F 3.7V 950mAh battery.

4.1.1 Query matching

The file search procedure is specified in the content discovery protocol, especially the descriptive content discovery[1]. Briefly, a file search consists of three main tasks: (1) the exchange of control messages between the requester peer and the provider peer, (2) the query matching that happens locally on a provider peer to compute the list of files that match the search expression, and (3) the metadata transfer of matched files. Steps (1) and (3) are inherently influenced by several factors: the distance between peers, the peer density of the overlay, and the peers' mobility. Therefore, we focus on step (2) and investigate how the query matching scales with the number of locally shared files, the overall number of file tags, the number of matched files, and the length of the search expression. Another section will deal with the effect of peer density.

To investigate the scalability of the query matching with respect to the number of shared files and the overall number of file tags, we use one mobile phone, say DEVICE1, to initiate file searches. On the other mobile phone, say DEVICE2, we measure the time it takes to run the query matching on receipt of each "FileSearch" message. We run this experiment on the measured device with varying number of shared files – from 100 to 1000 files, and varying number of tags per file – from 3 to 12 tags per file. For this experiment, we always tag the files on DEVICE2 (the measured device) in such a way that none will match searches coming from DEVICE1. This allows us to eliminate the overhead of caching matched files, and to consider only the effect of the number of shared files and the number of tags.

The second experiment is similar to the one above, except that this time, we choose search expressions from DEVICE1 that will yield varying number of hits on DEVICE2. This

[1] The identity-based discovery and the opportunistic discovery are really just precursors of file download

allows us to measure the scalability of the query matching with respect to the number of matched files.

The last experiment is similar to the second one, except that this time, we specify multi-word search expressions of varying number of words – specifically from 2 to 5 words; the multi-word search expressions are selected from DEVICE1 in such a way that each word in an expression will match a number of files on DEVICE2.

In all the above experiments, we measure the time it takes DEVICE2 to run the query matching on receipt of each file search.

4.1.2 Energy Consumption

To measure the energy consumption of a phone running *Macs*, we use the Nokia Energy Profiler (Nokia, 2009) tool. We are interested in standby energy consumption, which is the amount of energy consumed when the application runs in the background without user interaction. We think this is an important measurement because the standby mode represents the bulk of a phone's operational time. Nokia Energy Profiler (NEP) computes the power consumption of a phone by reading the built-in voltage meter once every 10s, and the current meter periodically (by default every 250ms). We configure NEP to read the current meter every 1s, in order to minimize the measurement activity overhead. To compute the amount of energy consumed by a peer, we multiply the power consumption reported from NEP by the length of time of an experiment.

We specifically run two experiments. In the first one, we use NEP to profile the power consumption of a single phone running *Macs* for an hour; at the end of the hour, we compute the energy consumption as described above. Because only one phone is running, we guarantee that

no communication will occur; therefore, this first experiment captures cases in which a peer is isolated – i.e. there is no other peer to interact with.

In the second experiment, we use the two phones as follows:

- On one phone, say DEVICE1, we modify the logic of *Macs* to automatically broadcast a file search after every *x* seconds; the value of *x* is randomly chosen between [1s, 300s), and the search expression is specified to a fixed string, namely "test". We keep track of all sent messages in a file on disk.
- On the other phone, say DEVICE2, we run the standard version of *Macs* and tag a shared file with the value "test". This ensures that all file searches received from the other phone result in hits (remember from section 3.2.4.1 that in this case, a lot messages are exchanged to transfer metadata of the matched file). We also keep track of all received messages in a file on disk.

This second experiment is run for an hour. At the end of the hour, we compute the energy consumption on DEVICE2. This experiment captures cases in which a peer is not isolated, but is involved in communication with other peers. By randomly broadcasting file searches from DEVICE1, we emulate a real environment in which multiple peers may be querying the network. Since we keep track of sent and received messages, we are able to compute the percentage of messages sent by DEVICE1 that were received on DEVICE2.

Remember that the energy management of *Macs* consists of dividing the execution time into intervals of equal length. During a time interval, a peer's wireless interface can either be on or off (see section 3.2.6 for more detail). Let us call *T*, the length of a time interval in seconds; the following table describes peer configurations used in the aforementioned experiments:

Table 4-1: Peer configurations for the measurement of energy consumption

Peer configuration	*Description*
Naïve	The peer runs a modified version of *Macs* that do not implement energy management – i.e. the wireless interface is always on
Macs with T=30	The peer runs the standard version of *Macs* with energy management, and a time interval T = 30 s.
Macs with T=60	Same as above, but with T = 60 s.

4.1.3 File Distribution

The content distribution protocol described in section 3.2.5 specifies how files are distributed in the overlay network. The current section aims for determining how fast this file distribution is, and how it is affected by the peer density and the mobility of the network. Due to a limited number of devices and hence the difficulty to setup a realistic test bed, we resort to simulation to reach our goal; we use ns-2 (McCanne *et al.*), a discrete event simulator for networking research. In what follows, we first describe our simulation environment, and then a description of the experiments run is given.

The simulated environment is any outdoor field populated with individuals moving at a pedestrian speed. An example of such environment can be students walking on a campus. In ns-2, we define a topology of 500m x 500m where peers move with a speed uniformly chosen from [0m/s, 1.5m/s] according to the random waypoint mobility model (Broch, *et al.*, 1998). In this mobility model, each peer starts from a different position and moves to a new randomly chosen destination with a constant speed. When a peer reaches its destination, it pauses for some time, and then starts moving again towards another randomly chosen destination. For our simulation, we define a 3-minute average pause time between movements. Each peer is equipped with an 802.11b wireless interface transmitting at a rate of 5.5Mb/s within a range of 100m. In all

experiments, we select the two-ray ground reflection model as the radio propagation model; this model considers both the direct path and a ground reflection path for radio waves transmission between peers. After setting up the simulation environment, we run two experiments.

The first one is the distribution of an 8MB file to varying number of peers. Only one peer has the file initially, and the others probe the network in order to download the file. In this experiment, the peer that initially owns the file is not counted; therefore, if we say the file is distributed to 5 peers, it means that there are actually 6 peers – one owning the file at the beginning. We run the experiment several times, each time with a different number of peers to distribute the file to. For each run, we measure the time it takes for all the peers to completely download the file. By varying the number of peers to which the file is distributed, this first experiment investigates the effect of peer density on file distribution.

The second experiment is the distribution of files of varying size to a fixed number of peers, namely 15. The purpose is to see how the file distribution scales with the file size. As with the first experiment, we run this experiment several times, each time with a file having a different size. We then measure for each run, the time it takes for the 15 peers to completely download a file.

A very important note is that for both experiments above, we run two variants. In *variant1*, a peer can simultaneously upload and download (see section 3.2.5.3). In *variant2*, the simultaneous upload/download is disabled. The following table summarizes the simulation environment.

Table 4-2: Simulation environment used for the performance evaluation of file distribution

Parameter	*Value*
Mobility model	Random waypoint
Peer pause time	3 min
Peer speed	[0m/s, 1.5m/s]

Peer transmission range	100m
Peer transmission rate	5.5Mb/s
Propagation model	Two-ray ground reflection
Topology	500m x 500m

4.2 Results and Discussion

4.2.1 Query matching

The query matching execution time as a function of the number of shared files and the number of file tags is shown in Figure 4-1.

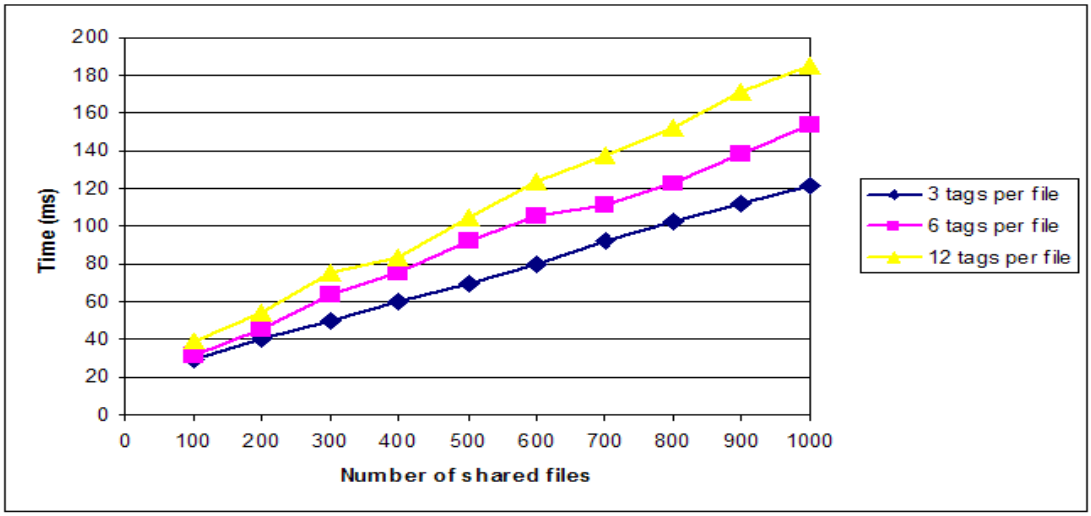


Figure 4-1: Query matching runtime for a non-matching search expression

The above figure reveals that the query matching execution time increases with the number of shared files. However, for the same amount of shared files, the execution time increases with the number of file tags. This makes sense because tags are the unit of comparison when determining matches. Therefore, the more tags the more comparison to perform. Despite this, the overall performance is satisfactory since the query matching still takes less than 190 ms

even on a database of 1000 shared files, each of which having 12 tags. To consider the effect of the number of matched files, Figure 4-2 shows the query matching runtimes for searches yielding varying number of matches.

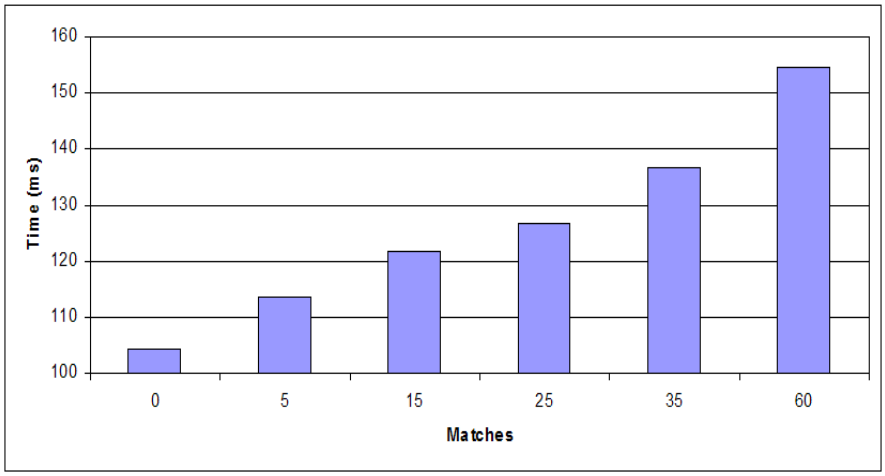


Figure 4-2: Query matching runtime vs. number of matches
The query matching is executed on a database of 500 shared files and 6000 tags.

From the above figure, one can see that the query matching runtime increases linearly with the number of matched files. This is in fact expected since in this case, the peer caches the matched files to prepare the transfer of their metadata. However, the performance is still satisfactory as it only takes about 152 ms to match 60 files. Another aspect of interest is how the length – number of words – of a search expression can affect the performance of the query matching. Therefore, Figure 4-3 plots the runtimes for multi-word search expressions.

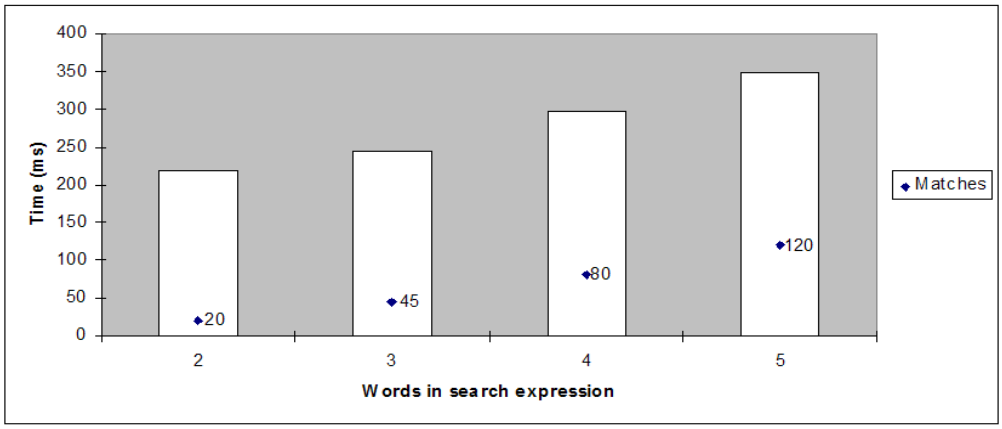


Figure 4-3: Query matching runtime vs. search expression length

Multi-word search expressions slow down the query matching. For instance while it takes 250 ms for a 3-word expression to return 45 matches (Figure 4-3), it only takes 152 ms for a single-word expression to return up to 60 matches (Figure 4-2). Multi-word search expressions take longer because the query matching runs against each word in an expression to rank the matched files as described in section 3.2.4.1.2. Nevertheless, the performance is still acceptable as it takes about 350 ms for a peer to match a 5-word search expression resulting in 120 matches, as shown in Figure 4-3. Besides, since mobile phones usually provide small-size keyboards, we argue that search expressions of more than 4 words will be very rare in real usage scenarios.

Remember that in section 3.3.1 we alluded to the fact that the database of *Macs* is non-1NF compliant. But, this is purposefully done as an optimization effort. To show the performance gain of our non-1NF design over a normalized design as described in section 3.3.1, Figure 4-4 plots the query matching runtimes of the two approaches.

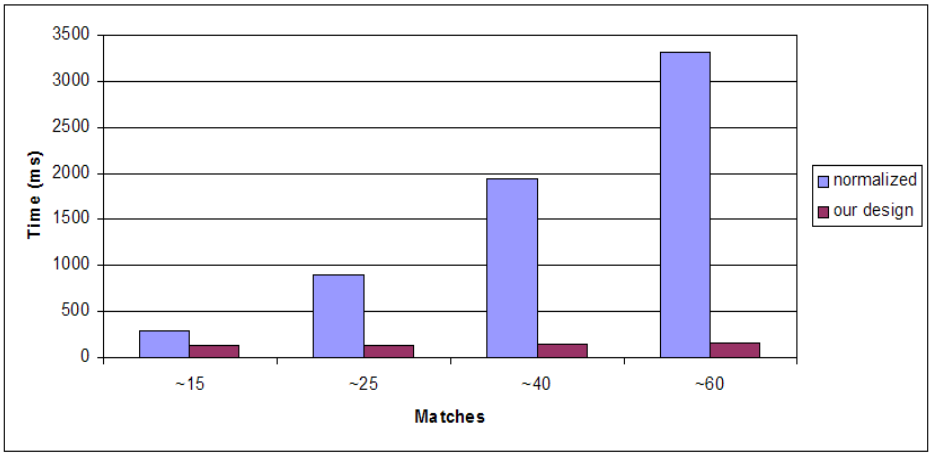


Figure 4-4: Query matching runtime: normalized database vs. our design

The performance gain of our design is clear; a normalized database would require unacceptable query matching runtimes as the number of matched files increases. This is due to the fact that a normalized database will have to use joined queries to return the list of matched files, the file tags and the other file attributes being in separate tables.

4.2.2 Energy Consumption

Figure 4-5 plots the energy consumption of *Macs* for the different peer configurations as defined in Table 4-1 above.

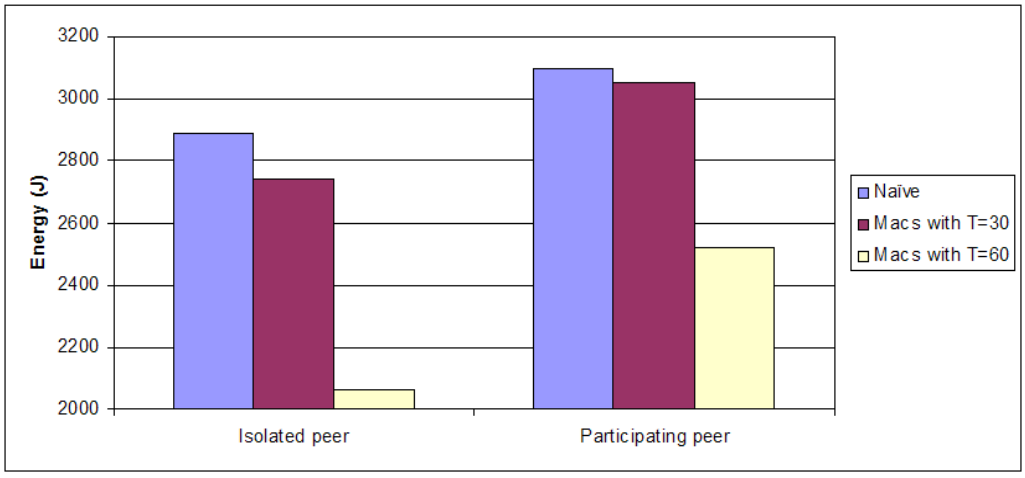


Figure 4-5: Energy consumption for different peer configurations

As one can see in the above figure, *Macs* with energy management provides the smallest energy consumption. For instance, when a peer is isolated (i.e. there is no message transmission), *Macs* with energy management and 60-second time interval yields a save of energy of 29% compared to an approach keeping the wireless interface always on. This performance gain is however less important when a peer is involved in communication (participating peer), because the length of time spent in sleep mode is reduced. But even when a peer is involved in communication, *Macs* still consumes less energy compared to what an isolated peer with the naïve approach would consume. Another interesting finding from the above figure is the effect of the time interval length, T, on energy consumption. It appears that smaller values of T results in lower save of energy. Therefore, from an energy point of view, it is beneficial to choose T big.

To show the effects of energy management on the efficiency of a *Macs* P2P overlay, Figure 4-6 adds to the plot of energy consumption, the percentage of messages lost due to our energy management implementation.

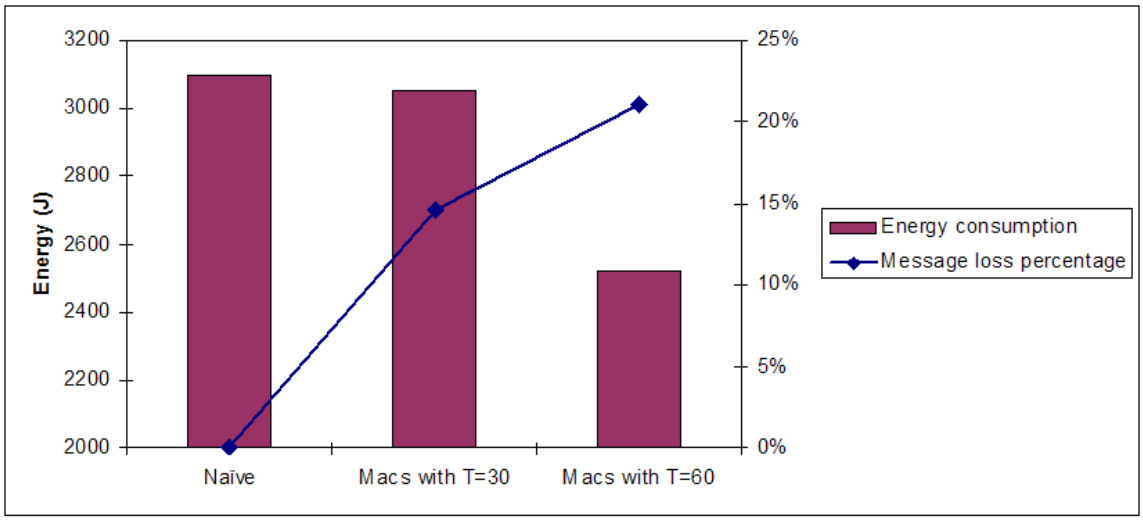


Figure 4-6: Effects of energy management on communication efficiency

As the above figure shows, our energy management has a negative impact on the communication efficiency of peers. However, for T = 60, we save 19% of energy for only 21% messages lost. Another interesting finding from the above figure is that, while it is beneficial to choose high values for T from an energy-consumption point of view (see Figure 4-5), the reverse is true from a communication point of view. Basically, a smaller value of T will yields a lower message loss percentage. This is understandable because when the time interval is long, a peer switching to sleep mode will remain unreachable for a long period of time and likely miss incoming messages.

4.2.3 File Distribution

Figure 4-7 plots the time it takes to distribute an 8MB file to several number of peers. Remember from section 4.1.3 that *Variant1* denotes the fact that a peer in the network can simultaneously upload and download chunks, whereas *Variant2* precludes that possibility.

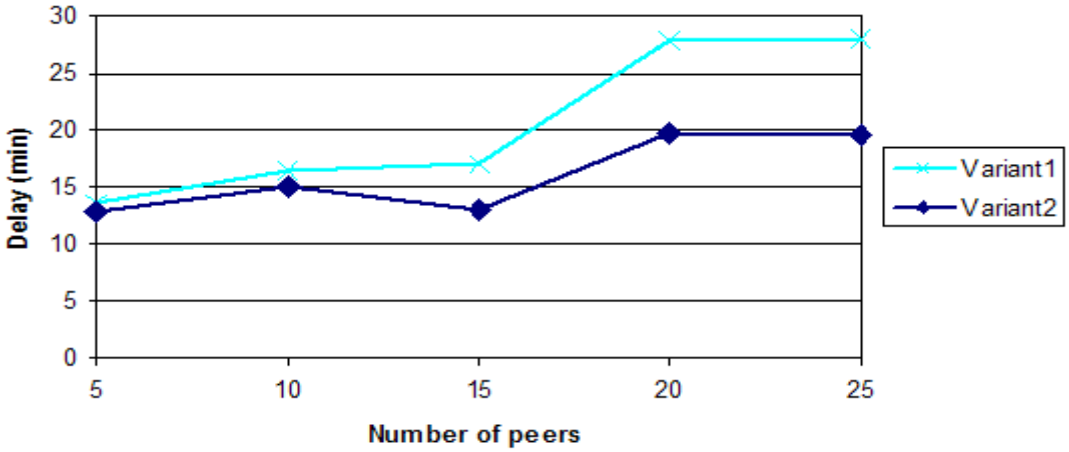


Figure 4-7: Delay for distributing an 8MB file to varying number of peers
Only one peer has the file initially, and that peer is not counted.

The above graph reveals that *Variant2* outperforms *Variant1*, which means it is better to avoid simultaneous upload/download. This is due to the half-duplex nature of 802.11b, making it less efficient for a peer to simultaneously send and receive data, especially in a dense network. As the graph shows, the difference between the two variants is intensified as the number of peers in the overlay network increases. Another interesting fact from these results is that the fraction of distribution delay to number of peers decreases linearly with the number of nodes. This implies that the file distribution performs better in dense networks. For instance, while it takes about 13 minutes to distribute the file to 5 moving peers, it only takes 20 minutes to distribute the same file to 25 peers, hence an average download time of 48 s per peer. Although Figure 4-7 shows that the file distribution delay decreases in a network of 15 peers for *variant2* (compared to 10 peers), the reader should not interpret it as an inconsistency, but as the result of the random mobility pattern used in the experiment. It happens that the mobility pattern file used in this particular case keeps the peers close to each other as they move.

Figure 4-8 plots the delay for distributing files of varying sizes to 15 mobile peers.

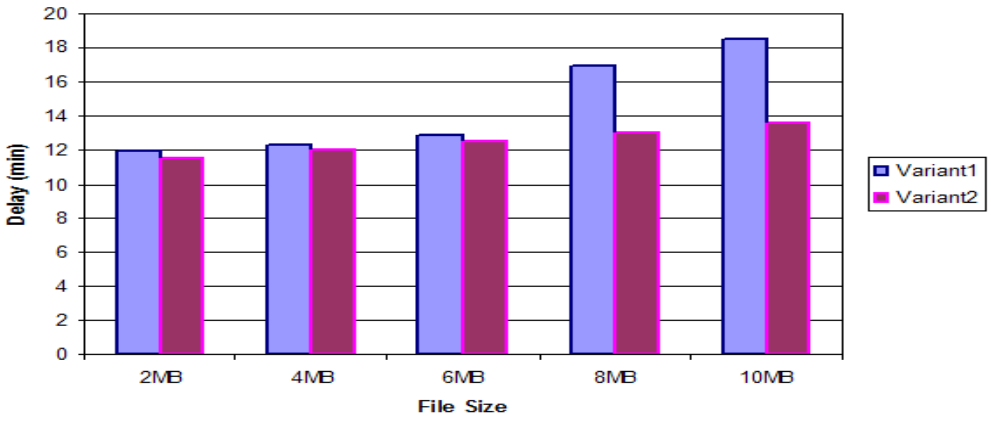


Figure 4-8: Distribution delay of varying-size files in a network of 15 moving peers

The graph confirms the statement that *variant2* outperforms *variant1*. Furthermore, it reveals that the file distribution delay increases very slightly with the size of the file for *variant2*. For instance, the overhead of distributing a 10MB file compared to a 2MB is only 2 minutes.

Chapter 5 Conclusion

5.1 Summary

In this work, we have explored the concept of mobile content sharing for ad hoc environments. We first demonstrated that most existing content sharing solutions on mobile phones were limited due to the fact that they either require a lot of manual configuration from users – e.g. pairing two devices for data transfer over Bluetooth, or they require the existence of an infrastructure-based network, such as the internet or the mobile phone network. Our claim for mobile content sharing tailored to ad hoc environments was motivated by the fact that, mobile phones are increasingly feature-rich and used by many people to self-generate content (pictures, video, etc.); besides, mobile internet is not always available (or at least affordable).

We then studied the feasibility of the concept of ad-hoc-based mobile sharing through a practical approach, consisting of developing a working prototype application. The prototype, *Macs*, is an unstructured, decentralized mobile P2P file sharing application for ad hoc environments that uses existing tools and provides capabilities that many internet users are familiar with, namely tagging, and keyword-based searching. Users tag the files they want to share, and they can discover content in the vicinity by specifying advanced keyword-based search options, such as “list of files that match any word or all words in a search expression”. Using the prototype, we carried out an extensive performance evaluation and found that ad-hoc-based content sharing is feasible on mobile devices. Specifically, the results of our research can be summarized as follows:

- Advanced keyword-based searching is efficiently possible on mobile P2P systems for ad hoc environments, though the time required to match local files on a device, slightly increases with the number of shared files and the number of tags.
- The density of mobile devices in the network positively contributes to the efficiency of file distribution.
- Energy consumption of wireless communication is a limiting factor of mobile P2P content sharing for ad hoc networks; but, it is possible to mitigate this with appropriate power conservation techniques.
- An energy management strategy based on duty cycle may provide not inconsiderable save of energy, but this would also have an effect on the communication efficiency of a peer.

5.2 Future work

Our current work does not propose any mechanism to ensure consistency between the actual content of a shared file and its tags. A malicious user can tag a corrupt file with commonly used expressions to enable the discovery of that file by other peers. Requester peers could then end up downloading files that are not related to their needs. Preventing such issue from happening is a complex task, which can hardly be done in the absence of a central entity. Investigating this possibility in mobile P2P content sharing for ad hoc networks constitutes an area of future research.

On the other hand, due to the fact that wireless communication on mobile devices is energy-consuming, mobile content sharing applications for ad hoc networks can be deterrent to

users. Therefore, exploring incentive mechanisms is a worthwhile effort. However, our current work does not define any such mechanism yet. This is an item for future work.

References

Chetan Sharma Consulting (2010). ‘Sizing up the Global Mobile Apps Market’ [online]. Available at: http://www.chetansharma.com/Sizing_up_the_Global_Mobile_Apps_Market.pdf. Last accessed: June 6, 2010.

Balasubramanian, N., Balasubramanian, and A., Venkataramani, A. (2009). 'Energy consumption in mobile phones: a measurement study and implications for network applications', in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, pp. 280-293.

Broch, J., Maltz, D., Johnson, D., Hu, Y., and Jetcheva, J. (1998). 'A performance comparison of multi-hop wireless ad hoc network routing protocols' in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, Dallas, Texas, USA, pp. 85-97.

Cheshire, S., Aboba, B., and Guttman, E. (2005). 'Dynamic Configuration of IPv4 Link-Local Addresses', RFC 3927 [online]. Available at: http://www.ietf.org/rfc/rfc3927.txt. Last accessed: June 21, 2010.

Chlamtac, I. (2003). 'Mobile ad hoc networking: imperatives and challenges'. *Ad Hoc Networks*, 1(1): 13-64.

Cisco (2008). *Global IP Traffic Forecast and Methodology*, White Paper.

Conti, M., Gregori, E., and Turi, G. (2005). 'A cross-layer optimization of gnutella for mobile ad hoc networks' in *Proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing*, Urbana-Champain, IL, USA, pp. 343-354.

Ding, G., and Bhargava, B. (2004). 'Peer-to-peer file-sharing over mobile ad hoc networks' in *Proceedings of the 2nd IEEE annual conference on pervasive computing and communications workshops*, pp. 104-109.

Emule Project: MobileMule (No date). [online]. Available at: http://mobil.emule-project.net. Last accessed: June 8, 2010.

Fitzek, F. H., and Charaf, H. (2009). 'Mobile Peer-to-Peer Networks: An Introduction to the Tutorial Guide', in *Mobile Peer to Peer (P2P)*, ed. by Fitzek, F.H., and Charaf, H.: John Wiley & Sons, Ltd, pp. 3-18.

Flinn, J., and Satyanarayanan, M. (1999). 'Energy-aware adaptation for mobile applications', *ACM SIGOPS Operating Systems Review*, 33(5): 48-63.

The Gnutella Protocol Specification v0.4 Document Revision 1.2 [online]. Available at: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf. Last accessed: June 13, 2010.

Hildrum, K., Kubatowicz, J., Rao, S., and Zhao, B. (2002). 'Distributed object location in a dynamic network', in *Proceedings of the 14th annual ACM symposium on parallel algorithms and architecture*, Winnipeg, Manitoba, Canada, pp. 41-52.

Howe, A. (2002). 'Napster and Gnutella: a comparison of two popular peer-to-peer protocols' [online]. Available at: http://members.tripod.com/ahowe_ca/pdf/napstergnutella.pdf. Last accessed: June 13, 2010.

ipoque (2009). 'ipoque Internet Study 2008/2009' [online]. Available at: http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009. Last accessed: June 8, 2010.

Juniper Research (2009). *Mobile Apps ~ Taking a bite of the apple*. White paper.

Kelényi, I., Csúcs, G., Forstner, B., and Charaf, H. (2007). 'Peer-to-Peer File Sharing for Mobile Devices', in *Mobile Phone Programming and its Application to Wireless Networking*, ed. by Fitzek, F.H., and Reichert, F., Dordrecht: Springer, pp. 311-324.

Kelényi, I., and Nurminen, J.K. (2008). 'Energy aspects of peer cooperation - Measurements with a mobile DHT system' in *Proceedings of cognitive and cooperative wireless networks workshop in the IEEE international conference on communications*, Beijing, China, pp. 164-168.

Kravets, R., and Zheng, R. (2005). 'On-demand power management for ad hoc networks' *Ad hoc networks*, 3(1): 51-68.

Liang, J., Kumar, R., and Ross, K. (2006). 'The KazaA overlay: a measurement study', *Computer Networks: The international Journal of Computer and Telecommunications Networking*, 50(6): 842-858

Liu, Y., Rahmati, A., Huang, Y., Jang, H., Zhong, L., Zhang, Y., and Zhang, S. (2009). 'xShare: supporting impromptu sharing of mobile phones', in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, Krakow, Poland, pp. 15-28.

MBIT.TV (2007). 'Mobile File Sharing' [online]. Available at: http://mbit.tv/index.jsp. Last accessed: June 8, 2010.

McCanne, S., and Floyd, S. 'The Network Simulator - ns-2' [online]. Available at: http://www.isi.edu/nsnam/ns. Last accessed: July 5, 2010.

Nokia (2006). 'S60 Platform SDKs for Symbian OS, for C++' [online]. Available at: http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html. Last accessed: June 28, 2010.

Nokia (2009). 'Nokia Energy Profiler' [online]. Available at http://www.forum.nokia.com/info/sw.nokia.com/id/324866e9-0460-4fa4-ac53-01f0c392d40f/Nokia_Energy_Profiler.html. Last accessed: July 19, 2010.

Nokia (2010). 'Qt - Cross-platform application and UI framework' [online]. Available at: http://qt.nokia.com/. Last accessed: june 28, 2010.

Papadopouli, M., and Schulzrinne, H. (2001). 'Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices' in *Proceedings of the 2nd ACM international symposium on mobile ad hoc networking & computing*, New York, NY, USA, pp. 117-127.

Pucha, H., Das, S., and Hu, Y. (2004). 'Ekta: An efficient DHT substrate for distributed applications in mobile ad hoc networks' in *Proceedings of the 6th IEEE workshop on mobile computing systems and applications*, Washington, DC, USA, pp. 163-173.

Rahmati, A., and Zhong, L. (2007). 'Context-for-wireless: context-sensitive energy-efficient wireless data transfer', in *Proceedings of the 5th international conference on Mobile systems, applications and services*, San Juan, Puerto Rico, pp. 165-178.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). 'A scalable content-addressable network', in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communication*, San Diego, California, United States, pp. 161-172.

Rowstron, A., and Druschel, P. (2001). 'Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems', in *Proceedings of the 18th IFIP/ACM international conference on distributed systems platforms (middleware 2001)*, Heidelberg, Germany.

Schollmeier, R. (2002). 'A Definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications', in *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, Washington, DC, USA.

Si, P., and Yu, R. (2009). 'Distributed sender scheduling for multimedia transmission in wireless mobile peer-to-peer networks', *IEE transactions on wireless communications*, 8(9): 4594-4603.

Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. (2001). 'Chord: A scalable peer-to-peer lookup service for Internet applications', in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communication*, San Diego, California, United States, pp. 149-160.

Stutzbach, D., Zhao, S., and Rejaie, R. (2007). 'Characterizing files in the modern Gnutella network', *Multimedia Systems*, 13(1): 35-50.

Tang, B., Zhou, Z., Kashyap, A., and Chiueh, T. (2005). 'An integrated approach for P2P file sharing on multi-hop wireless networks' in *Proceedings of the IEEE International Conference on wireless and mobile computing, networking and communication*, Montreal, Canada, pp. 268-274.

Wang, C., and Li, B. (2003). 'Peer-to-Peer overlay networks: a survey', *Technical Report*, Department of Computer Science, HKUST.

Wolfson, O., Xu, B., and Tanner, R. (2007). 'Mobile peer-to-peer data dissemination with resource constraints' in *Proceedings of the 2007 international conference on mobile data management*, Washington, DC, USA, pp. 16-23.

Zheng, R., Hou, J.C., and Sha, L. (2003). 'Asynchronous wakeup for ad hoc networks' in *Proceedings of the 4th ACM international symposium on mobile ad hoc networking & computing*, Annapolis, Maryland, USA, pp. 35-45.

Appendix A Message Formats

FileSearch

Bit offset	0	1	2	3	4	5	6	7
0	Search ID							
8								
16								
24								
32	Match option[1]							
40	Content type[3]							
48 …	Search expression							

QueryHit

Bit offset	0	1	2	3	4	5	6	7
0	Search ID							
8								
16								
24								
32	Hit ID							
40								
48								
56								
64								

MetadataRequest

Bit offset	0	1	2	3	4	5	6	7
0	Hit ID							
8								
16								
24								
32	Metadata count[2]							
40	Page index							
56								

InitMetadataTransfer

Bit offset	0	1	2	3	4	5	6	7
0	TCP port							
8								
16	Page count							
24								

FileProbe

Bit offset	0	1	2	3	4	5	6	7
0	Hash of probed file							
8								
16								
24								
32								
40 …	Indexes of chunks needed[4]							

FileProbeHit

Bit offset	0	1	2	3	4	5	6	7
0	Hash of probed file							
8								
16								
24								
32								
40 …	Indexes of available chunk							

FileDataRequest

Bit offset	0	1	2	3	4	5	6	7
0	File hash							
8								
16								
24								
32								

ReadyToSendFile

Bit offset	0	1	2	3	4	5	6	7
0	File hash							
8								
16								
24								
32								
40	TCP port							
48								

[1] Specify how to match the search expression. Three possible values: any word -1 -, all words -2 -, exact - 3 -.
[2] Maximum number of matched files whose metadata to return
[3] What file type to match: - 0 - all, - 1 - audio, - 2 - video.
[4] Represented as a bit field; a set bit means the chunk is available

Appendix B ns-2 Simulation Script

```
#===================================================================
# Initialization
#===================================================================
Phy/WirelessPhy set bandwidth_ 5.5Mb                ;# Data rate
Phy/WirelessPhy set Pt_ 0.1                         ;# Transmit power
Phy/WirelessPhy set RXThresh_ 9.55722e-10           ;# Receive power threshold
Phy/WirelessPhy set freq_ 2.442e9               ;# Data rate
Mac/802_11 set dataRate_ 5.5Mb                      ;# Rate for Data Frames
Mac/802_11 set basicRate_ 5.5Mb                     ;# Rate for Controle frames
Mac/802_11 set bandwidth_ 5.5Mb                     ;# Bandwidth
Mac/802_11 set RTSThreshold_ 2347                   ;# Rate for Control Frames
Application/ThesisControlApp set sche_interval_ 60 ;# Rate for broadcasting FPROBE

# ======================================================================
# Define options
# ======================================================================
set val(chan)         Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)        Phy/WirelessPhy            ;# network interface type
set val(mac)          Mac/802_11                 ;# MAC type
set val(ifq)          Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)           LL                         ;# link layer type
set val(ant)          Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)       50                         ;# max packet in ifq
set val(nn)           [lindex $argv 0]           ;# number of mobilenodes
set val(rp)           DumbAgent                  ;# routing protocol
set val(x)                    500                         ;# Side of the simulation grid
set val(pt)                   180
set val(sc)                   "scen_out/scen-$val(x)x$val(x)-$val(nn)-$val(pt)-1.5-1"   ;# node movement pattern
set val(stop)                 3600                        ;# simulation duration(seconds)

# ======================================================================
# Main Program
# ======================================================================

#
# Initialize Global Variables
#
set MESSAGE_PORT      5566                        ;# UDP port to listen to
set FILE_SIZE     [lindex $argv 1]                ;# File size (in chunks)
set ns_               [new Simulator]
set tracefile   traces/thesis_test232_${val(nn)}n_${FILE_SIZE}c
set tracefd           [open ${tracefile}.tr w]
set  nf [open thesis_test2.nam w]
set proid [lindex $argv 2]
# $ns_ use-newtrace
$ns_ namtrace-all-wireless $nf $val(x) $val(x)
$ns_ trace-all $tracefd

puts $tracefile
```

```
# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(x)

#
# Create God
#
create-god $val(nn)

#
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
#  to the channel.
#  Here two nodes are created : node(0) and node(1)

# configure node

      $ns_ node-config -adhocRouting $val(rp) \
                                 -llType $val(ll) \
                                 -macType $val(mac) \
                                 -ifqType $val(ifq) \
                                 -ifqLen $val(ifqlen) \
                                 -antType $val(ant) \
                                 -propType $val(prop) \
                                 -phyType $val(netif) \
                                 -channelType $val(chan) \
                                 -topoInstance $topo \
                                 -agentTrace ON \
                                 -routerTrace OFF \
                                 -macTrace OFF \
                                 -movementTrace OFF

            for {set i 0} {$i < $val(nn) } {incr i} {
                        set node_($i) [$ns_ node]
                        $node_($i) random-motion 0                ;# disable random motion
            }


# Define nodes initial position in nam
for {set i 0} {$i < $val(nn) } {incr i} {
            $ns_ initial_node_pos $node_($i) 20
}


# Attach UDP Message agents to nodes.
for {set i 0} {$i < $val(nn) } {incr i} {
            set udp_ag($i) [new Agent/MessagePassing]
            $node_($i) attach $udp_ag($i) $MESSAGE_PORT
}

# Attach TCP agents to nodes.
for {set i 0} {$i < $val(nn) } {incr i} {
            set src($i,0) [new Agent/TCP/FullTcp/ThesisTcp]
            set sink($i,0) [new Agent/TCP/FullTcp/ThesisTcp]
            set src($i,1) [new Agent/TCP/FullTcp/ThesisTcp]
            set sink($i,1) [new Agent/TCP/FullTcp/ThesisTcp]
```

```
        $ns_ attach-agent $node_($i) $src($i,0)
        $ns_ attach-agent $node_($i) $sink($i,0)
        $ns_ attach-agent $node_($i) $src($i,1)
        $ns_ attach-agent $node_($i) $sink($i,1)
}


# Attach a ThesisControlApp application to each UDP agent (and therefore, each node)
for {set i 0} {$i < $val(nn) } {incr i} {
        set app($i) [new Application/ThesisControlApp]
        $app($i) attach-agent $udp_ag($i)
        # By default, each node is looking for file 1
        $app($i) need 1 $FILE_SIZE
}
# Only node 3 owns the file
$app($proid) own 1 $FILE_SIZE


# Attach Traffic generator/consumer to each TCP agent
for {set i 0} {$i < $val(nn) } {incr i} {
        $app($i) attach-tcpsource $src($i,0)
        $app($i) attach-tcpsink $sink($i,0)
        $app($i) attach-tcpsource $src($i,1)
        $app($i) attach-tcpsink $sink($i,1)

        set sender($i,0) [new Application/TcpApp/ThesisDataSender $app($i)]
        set rcver($i,0) [new Application/TcpApp/ThesisDataRcver $app($i)]
        set sender($i,1) [new Application/TcpApp/ThesisDataSender "$app($i) 1"]
        set rcver($i,1) [new Application/TcpApp/ThesisDataRcver "$app($i) 1"]
}


# Define movement model
puts "... Loading movement file"
source $val(sc)


# Start application for each node.
for {set i 0} {$i < $val(nn) } {incr i} {
        $ns_ at [expr $val(pt) + 1*$i] "$app($i) start"
}


#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 3600.0 "$node_($i) reset";
}
$ns_ at $val(stop).0 "stop"
$ns_ at $val(stop).01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd nf
   $ns_ flush-trace
   close $tracefd
   close $nf
```

```
#    exec nam thesis1.nam &
    exit 0
}

puts "Starting Simulation..."
$ns_ run
```